

# Assignment – 1

## Logistic regression

Varshith Vootla

Ub It: 50365537

### Assignment Overview

We are implementing a logistic regression on diagnostic data given where the goal is to find if the patient has the diabetes or not. It is a 2-class problem which we are implementing with sigmoid as our hypothesis function. We use gradient descent to continuously update the theta and bias along with cost function to find the minimum on curve.

Also we implement the logistic regression with artificial neural networks with L1, L2 regularization and also we added the dropout layer and compare the accuracy and loss for training and validation data

### Dataset

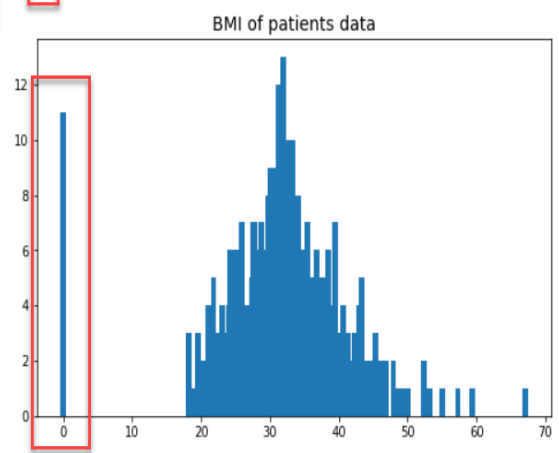
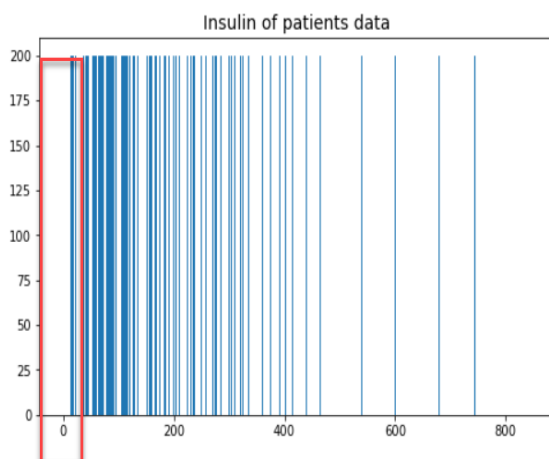
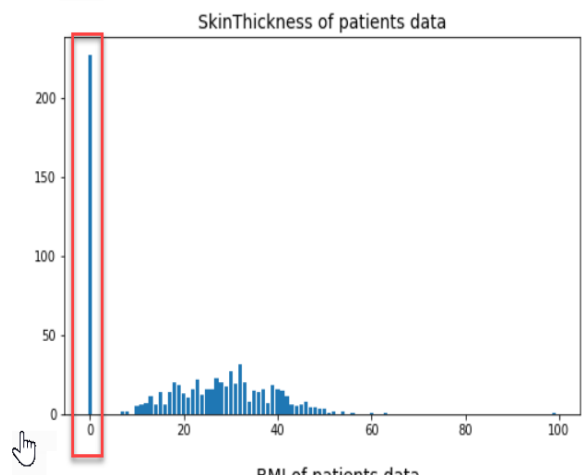
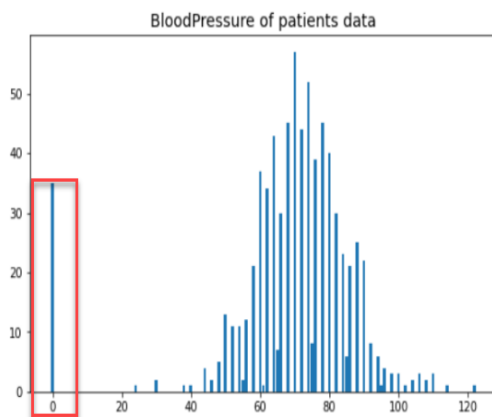
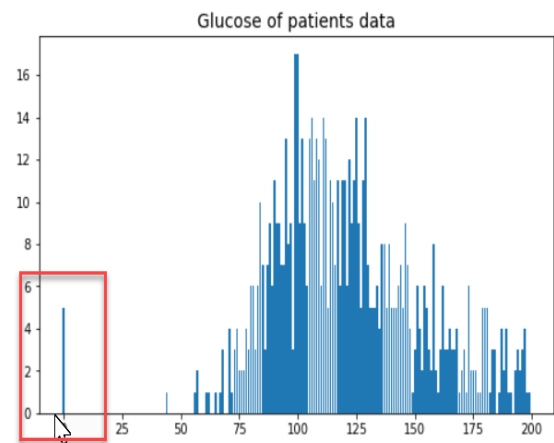
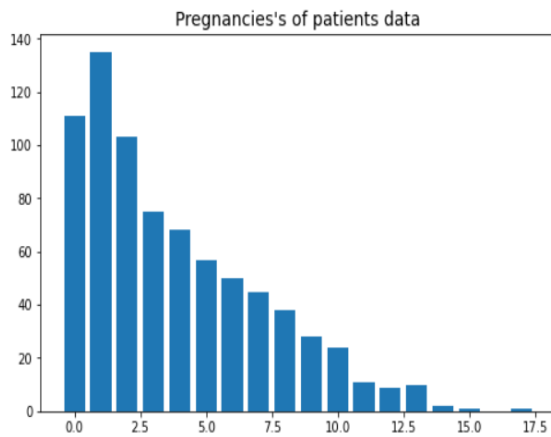
In total we have 768 samples of data and we divide the data into three sets randomly 60% for training data, 20% for each of validation and test data. This makes 461 samples for training and 154 each for both test and validation data

### Normalization

To proceed with logistic regression, we need to check the data for any junk values or unwanted data which is to be cleaned. If not cleaned the model will be trained with junk data which may give us erroneous results. To depict the junk data we plotted all the features with value counts.

When we see the data sample given some of the columns such as BloodPressure, SkinThickness, Insulin etc have 0 values which is not possible in the real life scenario of a patient. So we need to replace the data to normal values. It can be done either by replacing the 0 value with either mean or median. We are going to use mean for replacing the 0 values in

- Glucose
- SkinThickness
- BloodPressure
- BMI
- Insulin



All the red marked data shows 0 values in no. of samples for that particular columns which we replaced then with mean values of that particular columns.

### Feature scaling:

We also need all the features to be in same range. As we see pregnancies have 0 as data and insulin maximum levels are around 700. This needs to be concentrated in some particular range and is to be scaled around -1 to +1.

This is achieved by

Scaled value = value – mean/ Standard deviation

### Model and cost function

We have features = X

Target = Y

Weights =  $\theta$

$$h_{\theta}(x) = g(\theta^T x)$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

Cost function:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m y^i \log(h_{\theta}(x^i)) + (1 - y^i) \log(1 - h_{\theta}(x^i))$$

Loss function of Logistic Regression (m: number of training examples)

M – sample size

Y – target

H – hypothesis

## Gradient descent

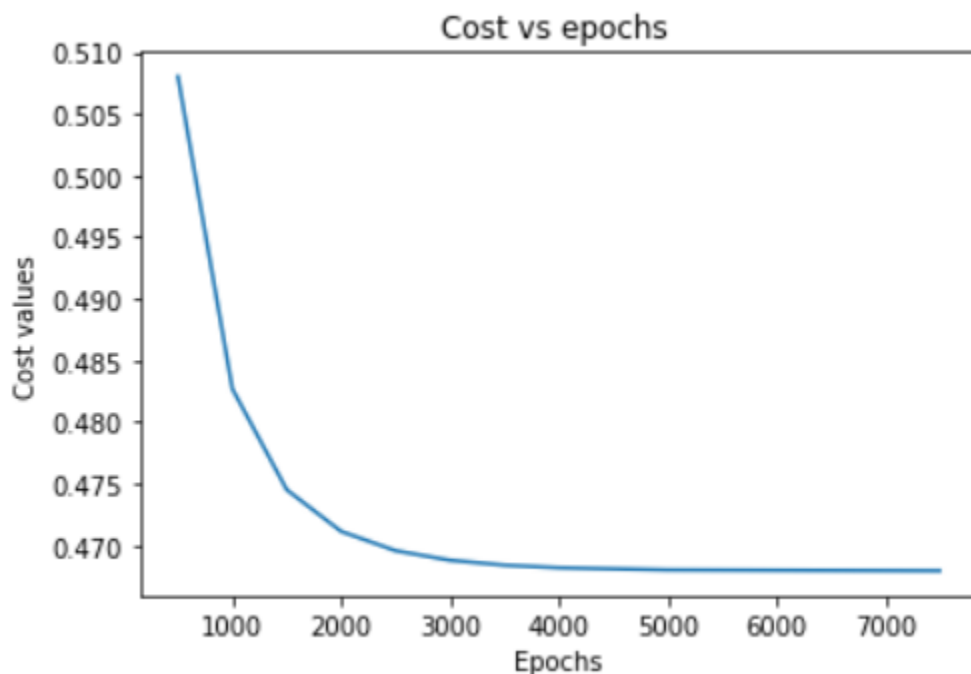
We use gradient descent to find the minimum weights or minima of decision boundary which can be the optimal solution to the problem where we find optimal weights or theta and bias.

Gradient descent assumes an initial weight vector with all zeroes and learning rate alpha with 0.01 value. It continuously loops all the iterations with initial theta, bias and learning rate which continuously produces new theta and bias values which we use in the cost function.

$$\Theta_j := \theta_j - \alpha \sum (h\theta(x(i)) - y(i))x(i)$$

`theta = theta - (alpha) * np.sum( loss * features))` (loss =  $h - y$ )

we simultaneously update theta and bias values and call loss function simultaneously for a set of iterations and when we plot the data we can see where the minimum point is achieved.



Above is visual representation of no. of iterations vs cost values.

After 5000 iterations the curve seems to be flattened for the data

We then get the theta and bias values which are optimal to be tested on test data

After running the gradient descent for our features vector and target for 5000 iterations we find the minimum cost

```
[new_theta_training,new_bias,updated] = gradientDescent(training_np_array,theta, bias, alpha, len(training_data),5000)
print("Initial Bias: ", bias)
print("Bias obtained after gradient descent: ", new_bias)

print("Learning rate alpha: ",alpha)

print("Initial theta: ", theta)
print("Theta obtained after gradinet descent",new_theta_training)

print("Cost: ",updated[0])

Initial Bias: 0.1
Bias obtained after gradient descent: -0.8176248374734875
Learning rate alpha: 0.01
Initial theta: [[0. 0. 0. 0. 0. 0. 0.]]
Theta obtained after gradinet descent [[ 0.46476784  1.17471517 -0.09459652  0.03069087 -0.31963842  0.63308455
 0.37585986  0.09233684]]
Cost: [0.46799978]
```

## Threshold

Threshold is a decision boundary where we assume the values to taken as class – 1 or class – 0

We assume 0.5 as threshold

If  $h \geq 0.5$  then class – 1

If  $h < 0.5$  then class – 2

## Accuracy

Now we use the test data and validation data to find the h or target with the theta and bias values obtained from gradient descent. We round the values obtained and then compare with our threshold to divide between classes.

We then compare the classes obtained with outcome column of the data given. And find the accuracy percentage.

Below are the details of accuracy for the data we used.

```

✓ [212] print("Training data accuracy is: ", findAccuracy(training_np_array,0.5,new_bias,new_theta_training))
      Training data accuracy is:  78.74186550976138

✓ [213] print("Validation data accuracy is: ", findAccuracy(validation_np_array,0.5,new_bias,new_theta_training))
      Validation data accuracy is:  79.87012987012987

✓ [214] print("Test data accuracy is: ", findAccuracy(test_np_array,0.5,new_bias,new_theta_training))
      Test data accuracy is:  77.27272727272727

```

**Accuracy for training data – logistic regression: 78.7%**

**Accuracy for validation data – logistic regression: 79.9%**

**Accuracy for test data – logistic regression: 77.2**

## Neural Network implementation with L1 and L2 regularization

Artificial neural network is implemented through tensorflow keras library

We are implementing 3 layers of neural network

- Input layer
- Hidden layer
- Output layer

```

model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(12, activation=tf.nn.relu),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(10, activation=tf.nn.relu,kernel_regularizer=tf.keras.regularizers.l2(0.01)),
    tf.keras.layers.Dense(1, activation=tf.nn.sigmoid)
])

```

For layer 1 and layer 2 we are using relu as activation function.

For the layer 3 sigmoid is used as activation function

To find the ideal no of neurons in the layer 1, no. of epochs, and batch size we run the hyper parameter tuning using tensor flow dashboard with hparams

## Hyperparameter tuning

```
HP_NUM_UNITS = hp.HParam('num_units', hp.Discrete([6, 8, 12]))
HP_EPOCHS = hp.HParam('epochs', hp.Discrete([32, 64]))
HP_DROPOUT = hp.HParam('dropout', hp.RealInterval(0.1, 0.2))
HP_BATCHES = hp.HParam('batches', hp.Discrete([20, 40]))
HP_LEARNING = hp.HParam('learning', hp.Discrete([0.01, 0.001, 0.1]))
HP_LAMBDA = hp.HParam('lambda', hp.Discrete([0.01, 0.05]))
```

We are tuning the above parameter.

### HP\_NUM\_UNITS

These are the no. of neurons or activations in the input layer and we run with 6, 8, 12 to find the ideal number

### HP\_EPOCHS

Epochs tuning with 32 and 64 iterations

### HP\_BATCHES

Batch size to be used for the ideal case

### HP\_LEARNING

Learning rate for adam optimizer

### HP\_LAMBDA

This is the penalty to be used for l1 and l2 regularizations

L1 and L2 regularization is implemented on the hidden layer of the artificial neural network

After running the model with all the parameters set we have accuracy and loss for all the possible combinations

Loss function used is binary\_crossentropy. It is equivalent to

$$J(\theta) = -1/m \cdot (y \log(h) + (1-y) \log(1-h))$$

## Tensor Dashboard

```
Starting trial: run-142  
{'num_units': 12, 'optimizer': 'adam', 'epochs': 64, 'batches': 40, 'learning': 0.1, 'dropout': 0.2, 'lambda': 0.01}  
15/15 [=====] - 0s 2ms/step - loss: 0.4053 - accuracy: 0.8482  
--- Starting trial: run-143
```

The screenshot shows the TensorBoard interface with the 'TABLE VIEW' selected. The table lists training results for various parameter combinations, sorted by accuracy in descending order. The left sidebar shows filters for metrics (Accuracy, Loss) and status (Unknown, Success, Failure, Running). The top navigation bar includes 'TensorBoard', 'SCALARS', 'HPARAMS', 'TIME SERIES', and an 'INACTIVE' status.

num_units	optimizer	epochs	dropout	batches	Accuracy	Loss
12.000	adam	64.000	0.10000	20.000	0.84816	0.40183
12.000	adam	32.000	0.20000	40.000	0.84816	0.42192
12.000	adam	64.000	0.20000	40.000	0.84816	0.40526
8.0000	adam	64.000	0.20000	40.000	0.83948	0.40311
12.000	adam	64.000	0.10000	40.000	0.83731	0.40735
12.000	adam	64.000	0.20000	40.000	0.83514	0.49535
6.0000	adam	64.000	0.10000	40.000	0.83080	0.42584
8.0000	adam	64.000	0.10000	40.000	0.83080	0.41323
12.000	adam	64.000	0.20000	40.000	0.82863	0.41546
8.0000	adam	64.000	0.20000	20.000	0.82863	0.42836
12.000	adam	64.000	0.20000	20.000	0.82646	0.40366
8.0000	adam	64.000	0.10000	20.000	0.82429	0.44336

After sorting the data with highest accuracy, we have the combination to use

12 – input activations

10 – Hidden layer neurons

Adam – optimizer

64 – epochs

40 – batch size

0.01 – Penalty for l1 and l2 regularization

0.1 – Learning rate

We have the highest possible accuracy of around 84%

Model is fit to run with above parameters and we can run it for validation and test data

Loss and accuracy for the divided data is with l1\_l2 regularization applied for the one and only hidden layer



We run the model with these parameters and check the accuracy and loss with 3 datasets we divided

### Accuracy Data for all datasets: (NN with l1 and l2)

```
▶ training_loss, training_accuracy = model.evaluate(x, y)
test_loss, test_accuracy = model.evaluate(X_test, y_test)
validation_loss, validation_accuracy = model.evaluate(X_validation, y_validation)

15/15 [=====] - 0s 2ms/step - loss: 0.4373 - accuracy: 0.8395
5/5 [=====] - 0s 2ms/step - loss: 0.4704 - accuracy: 0.8052
12/12 [=====] - 0s 2ms/step - loss: 0.4420 - accuracy: 0.8099
```

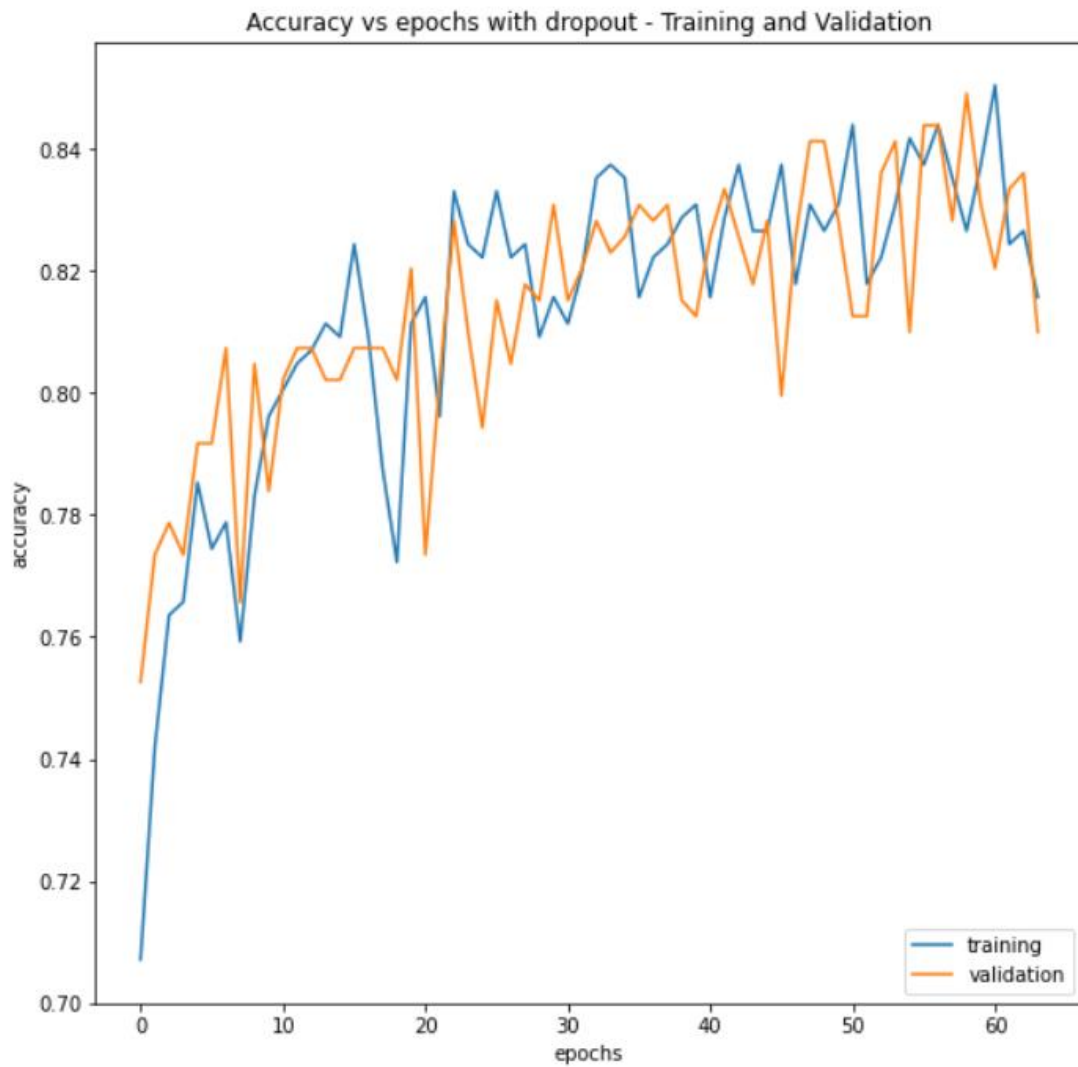
**Accuracy for training data – NN with l1 and l2: 83.9%**

**Accuracy for validation data – NN with l1 and l2: 80.5%**

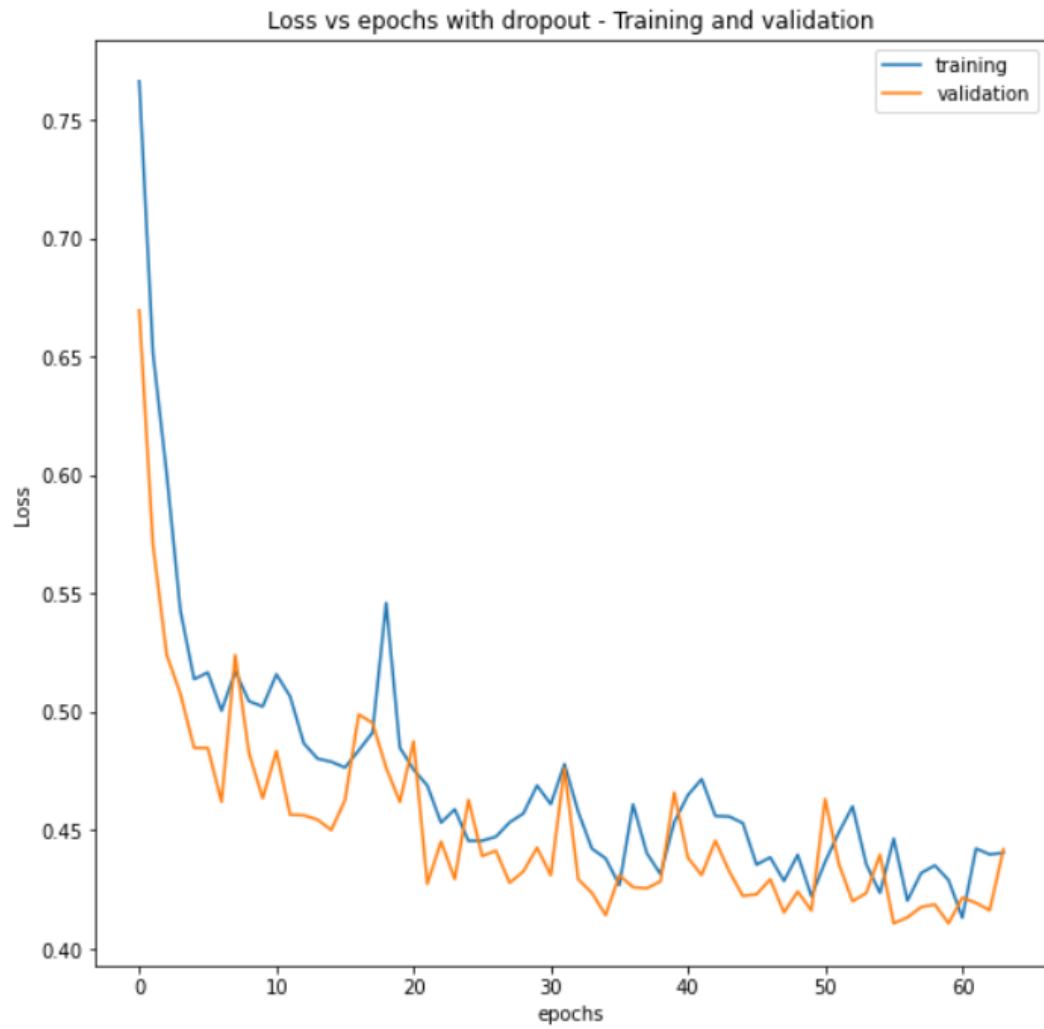
**Accuracy for test data – NN with l1 and l2: 81%**

## Training vs Validation – Accuracy:

s

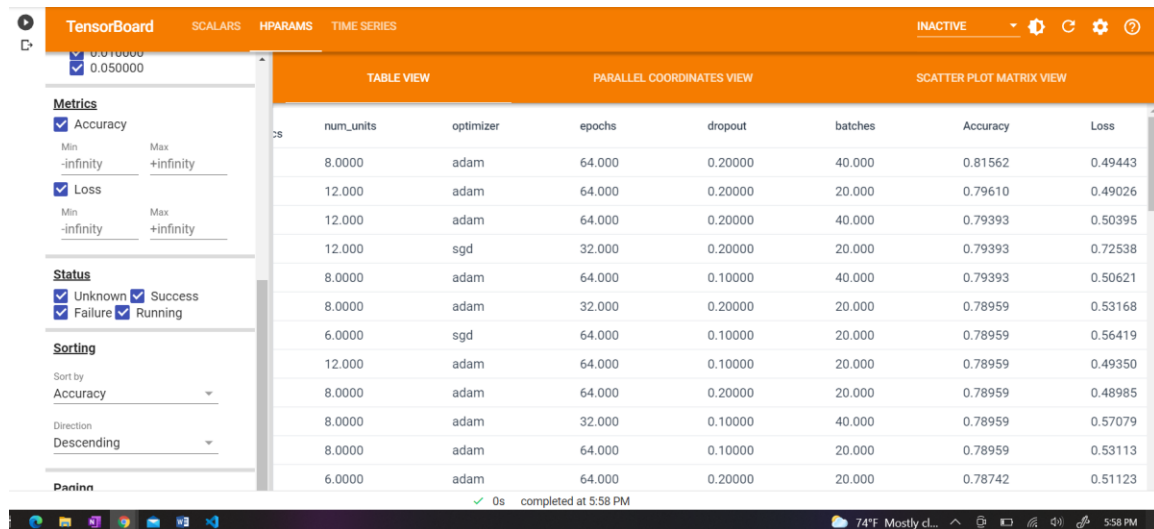


## Training vs Validation – Loss:



## Neural Network implementation with dropout and L1, L2

Also with L1 and L2 we are also adding dropout layer to the Artificial Neural Network with value of 0.2



The screenshot shows the TensorBoard interface with the 'TABLE VIEW' selected. The table displays training metrics for various configurations of num\_units, optimizer, epochs, dropout, and batches. The metrics include Accuracy and Loss. The left sidebar shows filters for Metrics (Accuracy, Loss) and Status (Unknown, Success, Failure, Running). The bottom status bar indicates '0s completed at 5:58 PM'.

num_units	optimizer	epochs	dropout	batches	Accuracy	Loss
8.0000	adam	64.000	0.20000	40.000	0.81562	0.49443
12.000	adam	64.000	0.20000	20.000	0.79610	0.49026
12.000	adam	64.000	0.20000	40.000	0.79393	0.50395
12.000	sgd	32.000	0.20000	20.000	0.79393	0.72538
8.0000	adam	64.000	0.10000	40.000	0.79393	0.50621
8.0000	adam	32.000	0.20000	20.000	0.78959	0.53168
6.0000	sgd	64.000	0.10000	20.000	0.78959	0.56419
12.000	adam	64.000	0.10000	20.000	0.78959	0.49350
8.0000	adam	64.000	0.20000	20.000	0.78959	0.48985
8.0000	adam	32.000	0.10000	40.000	0.78959	0.57079
8.0000	adam	64.000	0.10000	20.000	0.78959	0.53113
6.0000	adam	64.000	0.20000	20.000	0.78742	0.51123

### Accuracy for all data sets with dropout layer:

```
training_loss, training_accuracy = model.evaluate(X, y)
test_loss, test_accuracy = model.evaluate(X_test, y_test)
validation_loss, validation_accuracy = model.evaluate(X_validation, y_validation)
```

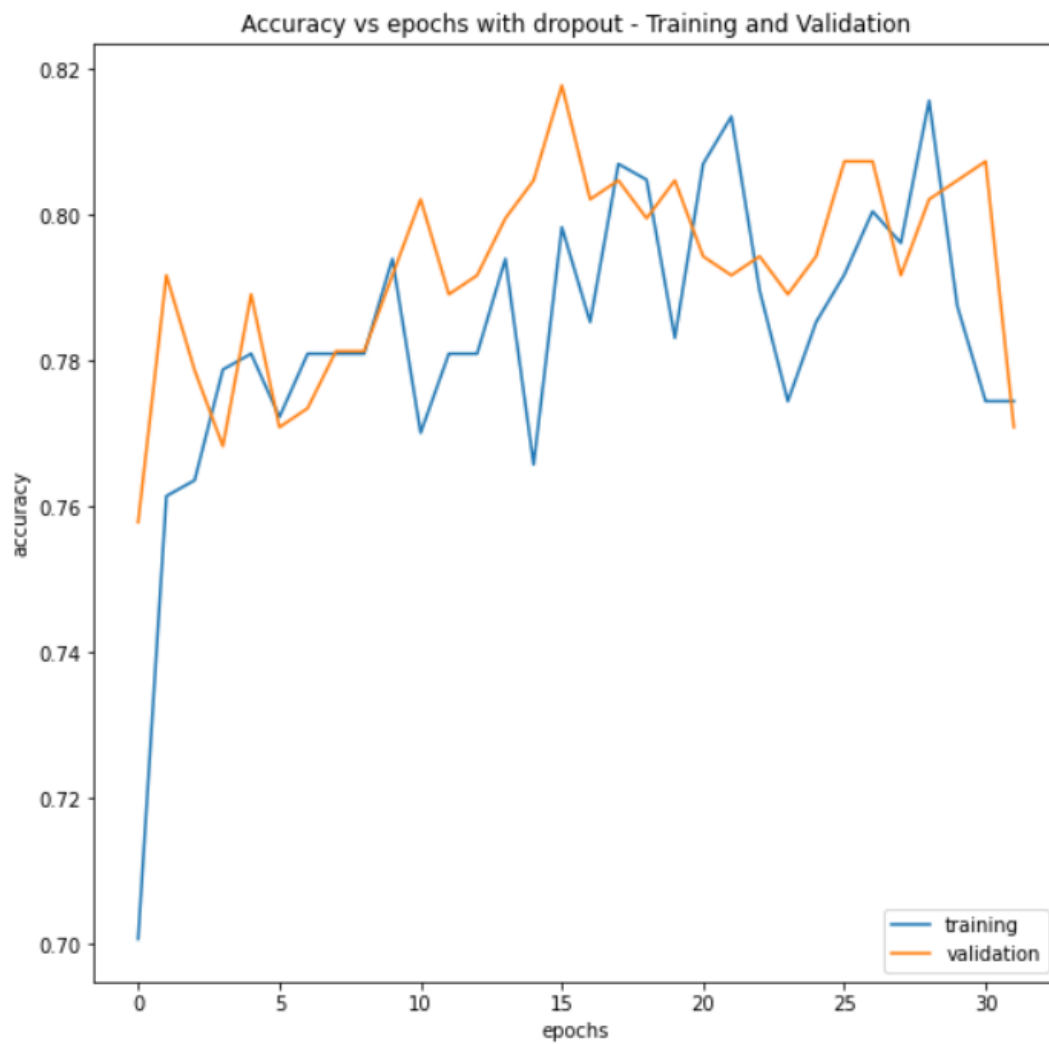
```
15/15 [=====] - 0s 1ms/step - loss: 0.4618 - accuracy: 0.8004
5/5 [=====] - 0s 2ms/step - loss: 0.4939 - accuracy: 0.7727
12/12 [=====] - 0s 2ms/step - loss: 0.4472 - accuracy: 0.7708
```

**Accuracy for training data – NN with l1 and l2 with dropout: 80%**

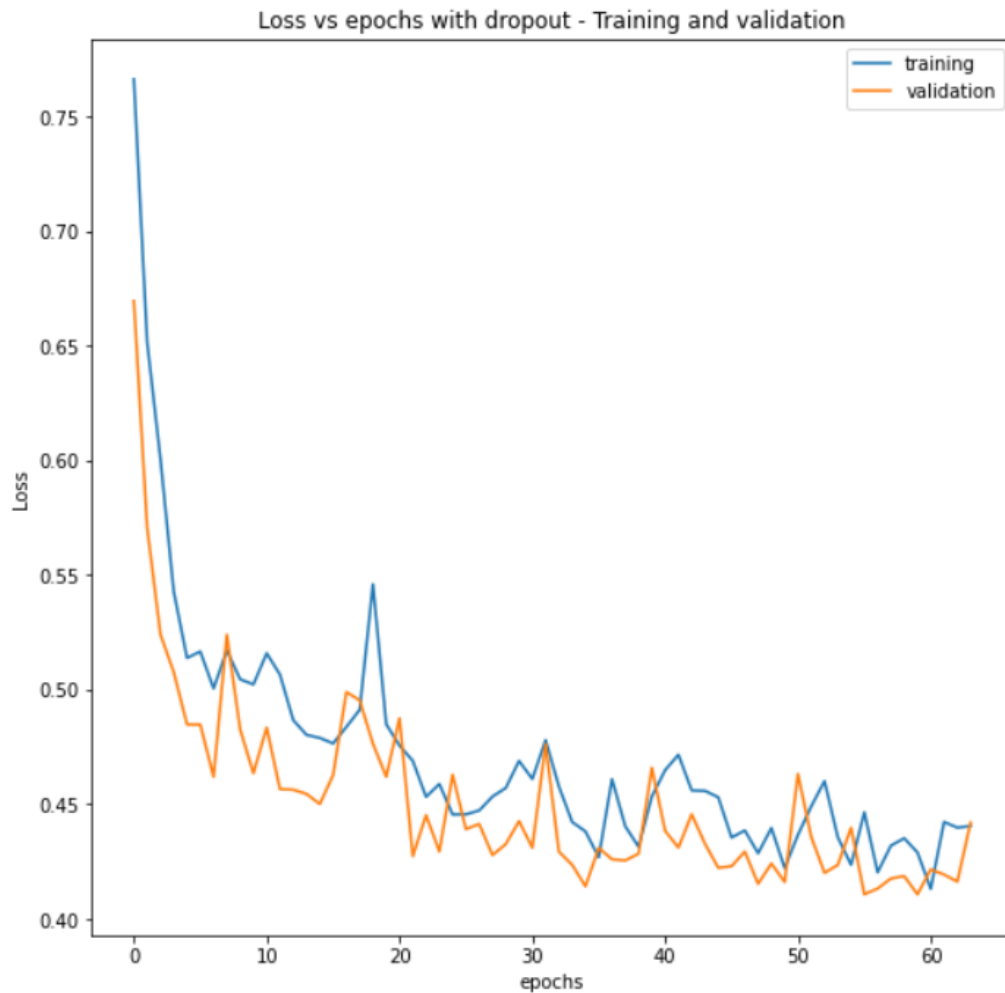
**Accuracy for validation data – NN with l1 and l2 with dropout: 77%**

**Accuracy for test data – NN with l1 and l2 with dropout: 77%**

## Training vs Validation – Accuracy – With dropout



## Training vs Validation – loss – with dropout



## Conclusion:

By comparing all the three models accuracy

	Logistic Regression	Neural Networks with L1 and L2	Neural Networks with dropout and L1, L2
Training data	78.7%	83.9%	80%
Test data	79.8%	80.5%	77%
Validation data	77.2%	81%	77%

Neural networks with L1 and L2 provides better generalization compared to other 2 models but it seems to overfitting the data since training accuracy is very high in comparison.

Dropout helped in removing the overfit but accuracy on validation didn't improve much in comparison with logistic regression

References:

[https://www.tensorflow.org/tensorboard/hyperparameter\\_tuning\\_with\\_hparams](https://www.tensorflow.org/tensorboard/hyperparameter_tuning_with_hparams) - For implementing hyperparameter tuning from tensorflow library